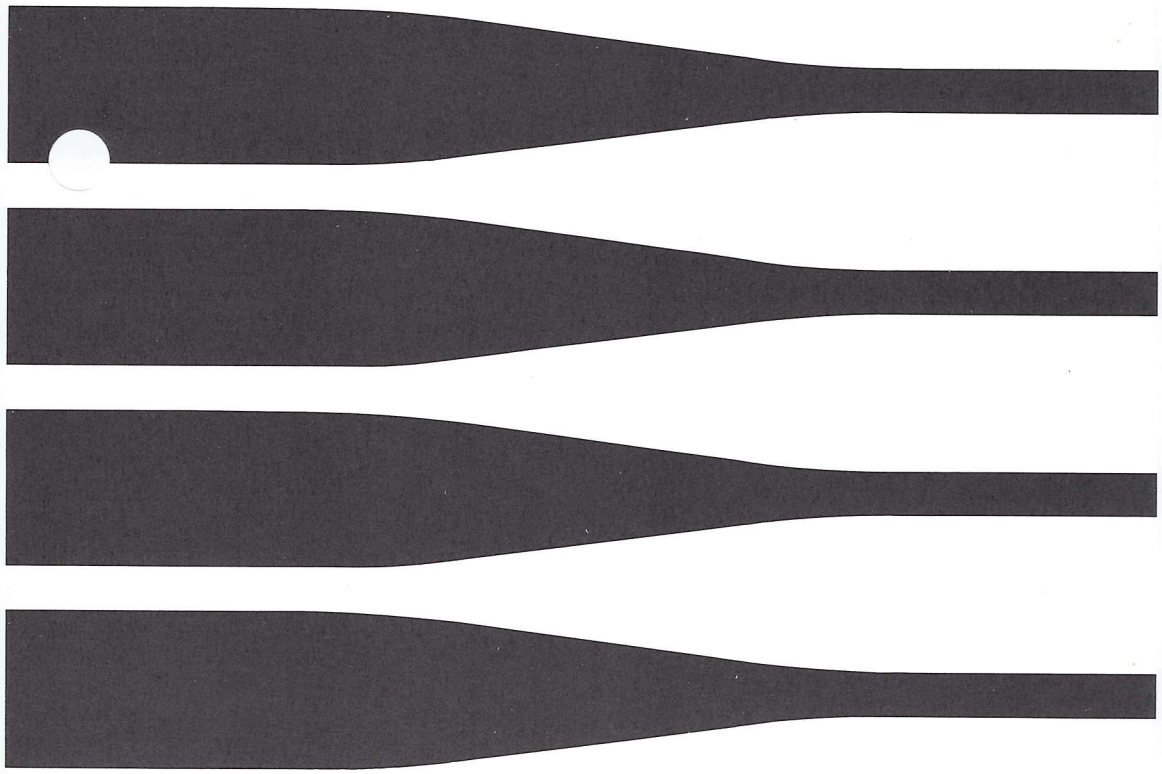
Issue 1
October 1984

AT&T 3B2 Computer UNIX™ System V Release 2.0 Security Administration Utilities Guide

Select Code
305-416

Comcode
403778368



TRADEMARKS

The following trademark is used in this manual:

- UNIX — Trademark of AT&T Bell Laboratories

NOTICE

The information in this document is subject to change without notice. AT&T Technologies assumes no responsibility for any errors that may appear in this document.

AT&T 3B2 COMPUTER
SECURITY ADMINISTRATION
UTILITIES GUIDE

Update Issue 1

December 10, 1984

This update inventory should be placed behind the *3B2 Computer Security Administration Utilities Guide* title page. This inventory identifies the pages that have been added or changed by the *3B2 Computer Security Administration Utilities Guide Update* (305-421), dated November 15, 1984.

Revised Page(s)	**Date**
ex manual pages	11/84
vi manual pages	11/84

NOTE

This Utilities Guide contains descriptive information and UNIX* System manual pages for the Security Administration Utilities. Since the Security Administration Utilities is only provided with 3B2 Computers sold in the United States, the manual pages for **crypt(1)** and **makekey** are not included in the *3B2 Computer UNIX System V User Reference Manual.* You can remove the manual pages for these commands from this Utilities Guide and file them in alphabetical order in the *User Reference Manual.*

Security Administration also makes the **-x** option available for encrypting files in **ed**, **ex**, and **vi**. The manual pages in the *User Reference Manual* for these commands do not include the **-x** option. Therefore, you should replace the **ed**, **ex**, and **vi** manual pages in the *User Reference Manual* with the manual pages from this Utilities Guide.

A UTILITIES binder is provided with the 3B2 Computer for you to keep the descriptive information from all the Utilities Guides together. Remove the descriptive information from the soft cover, place the provided tab separator in front of the title page, and file this material in the UTILITIES binder.

If you ordered extra copies of the *Security Administration Utilities Guide*, they should be left in their individual soft covers.

---

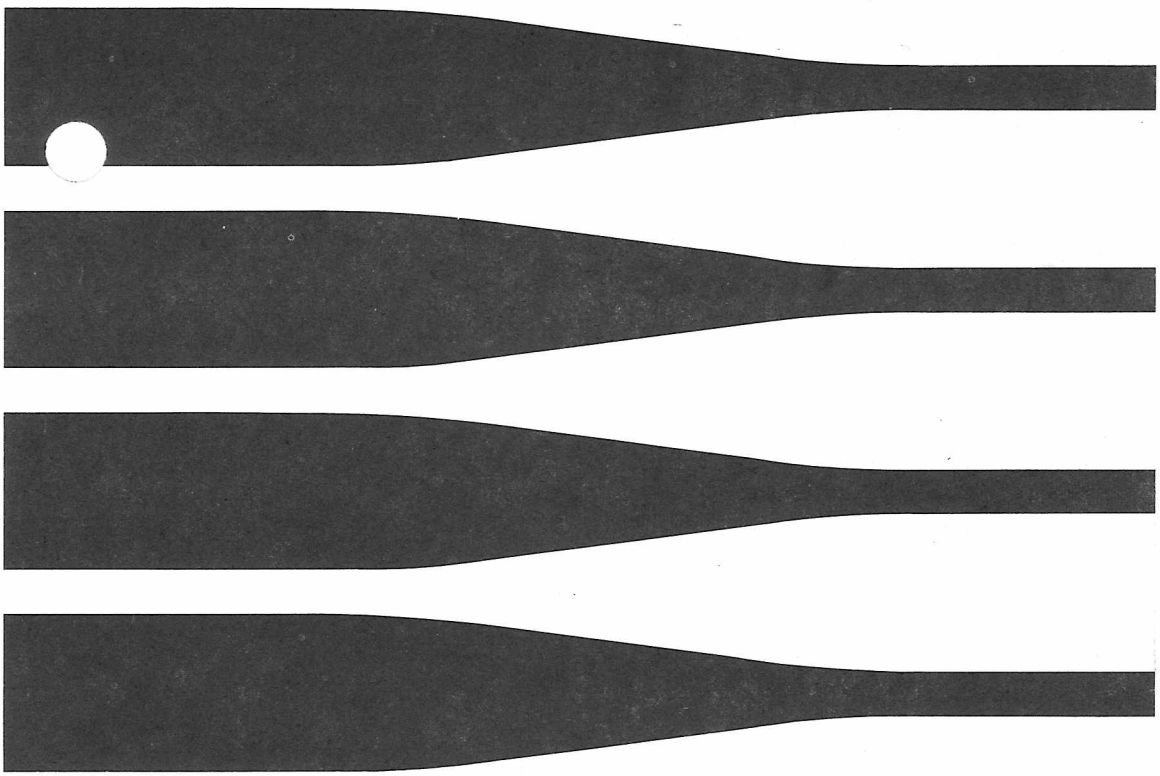
* Trademark of AT&T Bell Laboratories

Issue 1
October 1984

AT&T 3B2 Computer UNIX™ System V Release 2.0 Security Administration Utilities Software Information Bulletin

Select Code
305-353

Comcode
403778178



NOTE

This Software Information Bulletin (SIB) should be filed in the *3B2 Computer Owner/Operator Manual.* A tab separator, labeled " SOFTWARE INFORMATION BULLETINS," has been placed at the back of the *Owner/Operator Manual* in order to provide a convenient place for filing SIB's. Place the tab separator provided with this SIB in front of the title page and file this material behind the SOFTWARE INFORMATION BULLETINS tab separator in the *Owner/Operator Manual.*

SECURITY ADMINISTRATION SOFTWARE INFORMATION BULLETIN

INTRODUCTION

This Software Information Bulletin provides important information concerning the Security Administration Utilities. Please read this bulletin carefully before attempting to install or use these utilities.

The AT&T 3B2 Computer Security Administration Utilities contain additional security protection beyond that provided by login ID's, passwords, and permission modes. The Security Administration Utilities are part of the **UNIX*** System V Release 2.0 configuration provided with all 3B2 Computers.

SOFTWARE DEPENDENCIES

The Security Administration Utilities are dependent upon the Terminal Information Utilities. After the Security Administration Utilities has been installed, the Editing Utilities should not be installed.

If the Security Administration Utilities are removed, it is recommended that the Editing Utilities be reinstalled. When the Security Admininstration Utilities are installed, the Editor Utilities are overwritten with the crypt command. So, if the Security Administration Utilities are removed, the crypt command must also be removed. Therefore, for the editor to be fully functional, the Editor Utilities should be reinstalled.

---

* Trademark of AT&T Bell Laboratories

DOCUMENTATION

This Software Information Bulletin should be placed in the *3B2 Computer Owner/Operator Manual.*

The Security Administration Utilities commands are described in the *3B2 Computer Security Administration Utilities Guide.*

RELEASE FORMAT

Storage Structure

The Security Administraton Utilities command (CRYPT) is installed in the **/bin** directory.

The Security Administration Utilities command (MAKEKEY) is installed in the **/usr/lib** directory.

The Security Administration Utilities command (ed -x) is installed in the **/bin** directory.

The Security Administration Utilities command (vi -x) is installed in the **/usr/bin** directory.

System Requirements

The minimum equipment configuration required for the use of the Security Administration Utilities are 0.5 megabytes of random access memory and a 10-megabyte hard disk.

To install the Security Administration Utilities software there must be 115 free blocks of storage in **root** file system. Adequate storage space is checked automatically as part of the installation process. The installation process installs the utilities only if adequate storage space is available.

The Security Administration Utilities for the 3B2 Computer are distributed on one floppy disk.

Files Delivered

The Security Administration Utilities are delivered on a single floppy disk. The directory structure and files are as follows:

DIRECTORY	FILES
/bin	crypt ed
/usr/bin	vi
/usr/option	crypt.name ed.name
/usr/lib	expreserve exrecover exstrings_x makekey

UTILITIES INSTALL PROCEDURE

Use the standard software installation procedure described in the *3B2 Computer Owner/Operator Manual* for the installation of the Security Administration Utilities.

UTILITIES REMOVE PROCEDURE

Use the standard software remove procedure described in the *3B2 Computer Owner/Operator Manual* for the removal of the Security Administration Utilities.

CONTENTS

Chapter 1

INTRODUCTION

Chapter 1

INTRODUCTION

GENERAL

This guide describes command formats (syntax) and use of the Security Administration Utilities provided with your AT&T 3B2 Computer. The commands and procedures described in this guide provide additional security protection beyond that provided by login ID's, passwords, and permission modes. The additional protection is accomplished via the encryption capability provided with the Security Administration Utilities.

GUIDE ORGANIZATION

This guide is structured so you can easily find information without having to read the entire text. The remainder of this guide is organized as follows:

Chapter 2, "COMMAND DESCRIPTIONS," describes the formats (syntax) for each command in the Security Administration Utilities. The descriptions include the

purpose of the command, a discussion of the command syntax and options, and examples of using each command.

Appendix, "MANUAL PAGES," contains the Security Administration Utilities **UNIX*** System manual pages.

* Trademark of AT&T Bell Laboratories

Chapter 2

COMMAND DESCRIPTIONS

PAGE

Chapter 2

COMMAND DESCRIPTIONS

COMMAND SUMMARY

The Security Administration Utilities provide seven **UNIX** System commands. A summary of these commands is provided in Figure 2-1.

COMMAND	DESCRIPTION
crypt	This command is used to encode and decode files for security reasons. The crypt command reads from the standard input or keyboard, and writes on the standard output or terminal.
makekey	This command is used by the system to generate an encryption key.

Figure 2-1. Security Administration Utilities—Command Summary (Sheet 1 of 2)

COMMAND	DESCRIPTION
ed -x	This command is used to edit an existing file that has been encrypted or to create a new encryted file by using the **ed** text editor.
vi -x	This command is used to edit an existing file that has been encrypted or to create a new encryted file by using the **vi** text editor.
ex -x	This command is used to edit an existing file that has been encrypted or to create a new encrypted file by using the **ex** text editor.
edit -x	This command is used to edit an existing file that has been encrypted or to create a new encrypted file by using the **edit** text editor.
X	This command is used to encrypt a file while in the **(ed, edit, ex)** editor mode.

Figure 2-1. Security Administration Utilities—Command Summary (Sheet 2 of 2)

HOW COMMANDS ARE DESCRIBED

A common format is used to describe each of the commands. This format is as follows:

- **General:** The purpose of the command is defined. Any unique or special information about the command is also provided.
- **Command Format:** The basic command line format (syntax) is defined and the various arguments and options discussed.
- **Sample Command Use:** Example command line entries and system responses are provided to show you how to use the command.

In the command format discussions the following symbology and conventions are used to define the command syntax.

- The basic command is shown in bold type. For example: **command** is in bold type.
- Arguments that you must supply to the command are shown in a special type. For example: **command** *argument*
- Command options and arguments that do not have to be supplied are enclosed in brackets ([ ]). For example: **command** [*optional arguments*]
- The pipe symbol (¦) is used to separate arguments when one of several forms of an argument can be used for a given argument field. The pipe symbol can be thought of as an exclusive OR function in this context. For example: **command** [*argument1* ¦ *argument2*]

In the sample command discussions, user inputs and 3B2 Computer response examples are shown as follows:

```
This style of type is used to show system generated
responses displayed on your screen.

This style of bold type is used to show inputs
entered from your keyboard that are displayed on your
screen.

These bracket symbols, < > identify inputs from the
keyboard that are not displayed on your screen, such
as: <CR> carriage return, <CTRL d> control d, <ESC g>
escape g, passwords, and tabs.

This style of italic type is used for notes that
provide you with additional information.
```

Refer to the Appendix in this guide, the *3B2 User Reference Manual,* or the *AT&T 3B2 System Administration Utilities Guide,* as applicable, for **UNIX** System V manual pages supporting the commands described in this guide.

COMMAND DESCRIPTIONS

crypt — encode/decode files

General

The **crypt** command is used to encode and decode files for security purposes. This command must have a key (password) to encode the text. The same key also must be used to decode the file. The file is encoded in such a manner that no one can read the file unless the correct key is known.

If no key is given with the **crypt** command, then the **crypt** command will prompt a key from the terminal. While the key is being typed in on the terminal, the display and printer are disabled so the key cannot be seen. This is done for security reasons.

The choice of key security is the most vulnerable part of the **crypt** command. The best way to ensure key security is to select a complex, uncommon group of letters or numbers as the key. The key should be no more than 8 characters long. If the key is made to be a common or familiar entry, then someone may be able to "guess" your key.

If the same key is used to encrypt two or more files, be sure to decrypt them separately. If not, all files after the first named file to be decrypted will be garbled (in unreadable form).

A given file can be encrypted in the shell mode by using the **crypt** command or in the edit mode (ed/vi/ex/edit) by using the -x option or X command. The file will be encrypted the same way regardless of which mode you are in when you encrypt it. This file can be decrypted in the shell mode by using the **crypt** command, regardless of which method was used.

Command Format

The general format for the **crypt** command is as follows:

crypt *key <srcfile> desfile*

The *srcfile* is the file to be encrypted. The *desfile* is the name of the destination file for the encrypted text. The *srcfile* may be removed for security reasons.

Sample Commands

The following example shows how to encrypt a file named *secret* and have the encrypted text written to a file called *private*. The *key* is *246abcde*.

> ***Note:*** The following sample commands show the *srcfile* enclosed in brackets. These brackets and the corresponding *filename* must be entered by the user. This is an exception to the rule on "HOW COMMANDS ARE DESCRIBED."

```
$crypt 246abcde < secret > private<CR>
$
```

As previously discussed, the **crypt** command can be used without initially entering the key on the command line. In this case, the system will prompt the user to enter the key. The *key* is still *246abcde*.

```
$crypt < secret > private <CR>
ENTERKEY: <246abcde><CR>
$
```

Note: The key (246abcde) will not be seen on the display, it is shown here just for the example.

Any encrypted file can be displayed (in readable text) without permanently decrypting its contents. If the file is concatenated (cat), the display will be garbled (encrypted). In the previous example, the file *private* can be displayed with the *key (246abcde)* as shown in the following command line:

```
$crypt 246abcde < private<CR>
$
```

makekey — generate encryption key

General

The **makekey** command is used to generate the encryption *key* for the **crypt** command.

The **makekey** command reads 10 bytes from its input, and writes 13 bytes out on its output. The 13 byte output depends upon the 10 byte input in such a manner that it is intended to be very difficult to compute.

The 10 byte input consist of eight input bytes (input key) and two salt characters. The two salt characters are created from the set of digits, ., /, and upper and lower-case letters.

The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt characters and constitute the 13 byte output *key*.

With this command, there are no command formats or sample commands because the **makekey** command works internally with the Security Administration Utilities.

ed, edit, ex, vi — editors in which files can be encrypted and decrypted.

General

The **(ed, edit, ex, vi)** editor can be used to either edit an existing file that has been encrypted or to create a new encrypted file, by using the -x option. This option must have a key (password) to encrypt or decrypt the specified file. The same key that was used to encrypt a file must be used to decrypt it. The file is encrypted in such a manner that no one can read the file unless the correct key is known.

The X command is another way to encrypt a file while in the editor mode. This command works for the **(ed, edit, ex)** editor. This command requires a key (password) to encrypt the file. The same key that was used to encrypt the file also has to be used to decrypt it.

The choice of key security is the most vulnerable part of the Security Administration Utilities. The best way to ensure key security is to select a complex, uncommon group of letters or numbers as the key. The key should be no more than 8 characters long. If the key is a common or familiar entry, then someone may be able to "guess" your key.

A given file can be encrypted in the shell mode by using the **crypt** command, or in the edit mode **(ed/vi/ex/edit)** by using the -x option or X command. The file will be encrypted the same way regardless of which mode you are in when you encrypt it. This file can be decrypted in the shell mode by using the **crypt** command, regardless of which method was used.

Command Format

The command format for each of the editors (ed, edit, ex, vi) using the -x option is as follows:

ed [**-x**] *[filename]*
edit [**-x**] *[filename]*
ex [**-x**] *[filename]*
vi [**-x**] *[filename]*

The *-x* option is used to either edit an existing file that has been encrypted or to create a new encrypted file. The filename argument is the name of the file that is being created or edited.

Sample Command

ed Sample Command

The following example shows how to create an encrypted file called *secret*. The *key* is *135abcde*.

```
$ed -x secret<CR>
Enter file encryption key: <135abcde><CR>
?secret
```

Note: The key (135abcde) will not be seen on the display, it is shown here just for an example.

To enter text, you must be in the append mode. To enter the append mode, type the following:

```
a<CR>
(enter text)
now is the time for all good men.....
now is the time for all good men.....
```

To exit the append mode type the following.

.<CR>

Now write the contents of the buffer to disk by entering the following command.

w<CR>

To exit from the editor mode, type the following command.

q <CR>

Now that you are in the shell (prompt $), use the **cat** command to print the contents of the file *secret*. You will observe that the file is encrypted.

To decrypt the file *secret*, you must use the **crypt** command. The same *key 135abcde* must be used.

The following example shows how to enter an existing encrypted file called *secret* and to print out the text (in readable form). For instructions on using the **ed** (editor), refer to the UNIX SYSTEM V USER GUIDE. The same *key 135abcde* must be used.

```
$ed -x secret<CR>
Enter file encryption key: <135abcde><CR>
40 (character count)
1,$p<CR>
now is the time for all good men.....
now is the time for all good men.....

Note: the key (135abcde) will not be seen
on the display, it is shown here just for the
example.
```

To exit the editor mode and return to the shell (prompt $), type the following command.

q<*CR*>

Caution: If you change the text in any way do not forget to write the buffer to hard disk, before leaving the editor mode.

The following is an example of printing a unencrypted file called private in the ed editor, then encrypting it with the X command.

```
$ed private<CR>
140 (character count)
1,$p<CR>
now is the time for all good....
now is the time for all good....
X<CR>
Enter file encryption key: <1abcd><CR>
w<CR>
140(character count)
q<CR>
```

Note: The key (1abcd) will not be seen on the display, it is shown here just for the example.

edit Sample Command

The following example shows how to create an encrypted file called *secret*. The *key* is *135abcde*.

```
$ edit -x secret<CR>
KEY: <135abcde><CR>
"secret" [Newfile]
:

Note: The key (135abcde) will not be seen
on the display, it is shwon here just for the
example.
```

To enter text, you must be in the append mode. To enter the append mode, type the following.

```
a<CR>
(enter text)
now is the time for all good men......
now is the time for all good men......
```

To exit the append mode, type the following.

.<*CR*>

Now write the contents of the buffer to disk by entering the following command.

w<*CR*>

To exit from the editor mode, type the following command.

q<*CR*>

Now that you are in the shell (prompt $), use the **cat** command to print the contents of the file *secret*. You will observe that the file is encrypted.

To decrypt the file *secret* you must use the **crypt** command. The same *key 135abcde* must be used.

The following example shows how to enter an existing encrypted file called *secret* and to print out the text (in readable form). For instructions on using the **edit** (editor) refer to the 3B2 Computer Editing Utilities Guide. The same *key 135abcde* must be used.

```
$ edit -x secret<CR>
KEY: <135abcde><CR>
"secret" n line, n characters
:
1,$p<CR>
now is the time for all good men......
now is the time for all good men......
```

Note: The key (135abcde) will not be seen on the display, it is shown here just for the example.

To exit the editor mode and return to the shell (prompt $), type the following command.

q<*CR*>

Caution: If you change the text in any way do not forget to write the buffer to hard disk, before leaving the editor mode.

The following is an example of printing a unencrypted file called private in the edit editor, then encrypting it with the X command.

```
$edit private<CR>
" private"  n lines, n characters
:1,$p
text
now is the time for all....
now is the time for all....
:X<CR>
Entering encrypting mode. key: <123bcd45><CR>
:at end-of-file
:w<CR>
" private"  n lines, n characters
:q<CR>
```

Note: The key (123bcd45) will not be seen on the display, it is shown here just for an example.

ex Sample Command

The following example shows how to create an encrypted file called *secret*. The *key* is *135abcde*.

```
$ ex -x secret<CR>
KEY: <135abcde><CR>
"secret" [Newfile]
:
```

Note: The key (135abcde) will not be seen on the display, it is shown here just for the example.

To enter text, you must be in the append mode. To enter the append mode, type the following.

```
a <CR>
(enter text)
now is the time for all good men......
now is the time for all good men......
```

To exit the append mode, type the following.

.<*CR*>

Now write the contents of the buffer to the disk by entering the following command.

w<*CR*>

To exit from the editor mode, type the following command.

q<*CR*>

Now that you are in the shell (prompt $), use the **cat** command to print the contents of the file *secret*. You will observe that the file is encrypted.

To decrypt the file *secret*, you must use the **crypt** command. The same *key 135abcde* must be used.

The following example shows how to enter an existing encrypted file called *secret* and to print out the text (in readable form). For instructions on using the **ex** (editor), refer to the 3B2 Computer Editing Utilities Guide. The same *key 135abcde* must be used.

```
$ ex -x secret<CR>
KEY: <135abcde><CR>
"secret" n line, n characters
:
1,$p<CR>
now is the time for all good men.....
now is the time for all good men.....
```

Note: The key (135abcde) will not be seen on the display, it is shown here just for the example.

To exit the editor mode and return to the shell (prompt $), type the following command.

q<*CR*>

Caution: If you change the text in any way, do not forget to write the buffer to hard disk before leaving the editor mode.

The following is an example of printing a unencrypted file called private in the ex editor, then encrypting it with the X command.

```
$ex private<CR>
"private" n lines, n characters
:1,$p<CR>
text
now is the time for all good men...
now is the time for all good men...
:X<CR>
Entering encrypting mode. key: <123abc><CR>
: at end-of-file
w<CR>
" private" n lines, n characters
:q<CR>
```

Note: The key (123abc will not be seen on the display, it is shown here just for the example.

vi Sample Command

The following example shows how to create an encrypted file called *private*. The *key* is *678abcde*.

```
$ vi -x private<CR>
key: <678abcde><CR>

Note: The key (678abcde) will not be seen
on the display, it is shown here just for the
example.
```

To enter text, you must be in the input mode. To enter the input mode, type the following.

```
<a>
(enter text)
now is the time for all good men.....
now is the time for all good men.....
```

To exit the input mode, depress the **ESC** key.

To write the buffer to disk and return to the shell, type the following command.

ZZ

Now that you are in the shell (prompt $), use the **cat** command to print the contents of the file *private*. You will observe that the file is encrypted.

To decrypt the file *private*, you must use the **crypt** command. The same *key 678abcde* must be used.

The following example shows how to enter an existing file called *private* and to print out the text (in readable form). For instruction on using the **vi** (editor), refer to the 3B2 Computer Editing Utilities Guide. The same *key 678abcde* must be used.

```
$ vi -x private<CR>
Key: <678abcde><CR>
(text)
now is the time for all good men.....
now is the time for all good men.....
```

Note: The key (678abcde) will not be seen on the display, it is shown here just for the example.

To exit the editor mode and write the buffer to disk, type the following command.

ZZ

Appendix

MANUAL PAGES

This appendix contains the **UNIX*** System Manual Pages for the Security Administration Utilities. Manual pages for the following commands are provided in alphabetical sequence.

crypt
ed -x
edit -x
ex -x
makekey
vi -x

For your convenience, the user manual pages for the Security Administration Utilities are provided alphabetically in both this guide and in the *3B2 User Reference Manual.* The yellow sheet provided in this manual describes your options for filing the manual pages as well as descriptive information.

NAME

crypt – encode/decode

SYNOPSIS

crypt [password]

DESCRIPTION

Crypt reads from the standard input and writes on the standard output. The *password* is a key that selects a particular transformation. If no *password* is given, *crypt* demands a key from the terminal and turns off printing while the key is being typed in. *Crypt* encrypts and decrypts with the same key:

```
crypt key <clear >cypher
crypt key <cypher | pr
```

Files encrypted by *crypt* are compatible with those treated by the editor *ed* in encryption mode.

The security of encrypted files depends on three factors: the fundamental method must be hard to solve; direct search of the key space must be infeasible; "sneak paths" by which keys or clear text can become visible must be minimized.

Crypt implements a one-rotor machine designed along the lines of the German Enigma, but with a 256-element rotor. Methods of attack on such machines are known, but not widely; moreover the amount of work required is likely to be large.

The transformation of a key into the internal settings of the machine is deliberately designed to be expensive, i.e., to take a substantial fraction of a second to compute. However, if keys are restricted to (say) three lower-case letters, then encrypted files can be read by expending only a substantial fraction of five minutes of machine time.

Since the key is an argument to the *crypt* command, it is potentially visible to users executing *ps*(1) or a derivative. The choice of keys and key security are the most vulnerable aspect of *crypt*.

FILES

/dev/tty for typed key

SEE ALSO

makekey(1), ed(1).
stty(1) in the *3B2 Computer System User Reference Manual.*

BUGS

If output is piped to *nroff* and the encryption key is *not* given on the command line, *crypt* can leave terminal modes in a strange state (see *stty*(1)).
If two or more files encrypted with the same key are concatenated and an attempt is made to decrypt the result, only the contents of the first of the original files will be decrypted correctly.

NAME

ed, red – text editor

SYNOPSIS

ed [–] [**–p** string] [**–x**] [file]

red [–] [**–p** string] [**–x**] [file]

DESCRIPTION

Ed is the standard text editor. If the *file* argument is given, *ed* simulates an *e* command (see below) on the named file; that is to say, the file is read into *ed*'s buffer so that it can be edited. The optional – suppresses the printing of character counts by *e*, *r*, and *w* commands, of diagnostics from *e* and *q* commands, and of the ! prompt after a *!shell command*. The **–p** option allows the user to specify a prompt string. If **–x** is present, an *x* command is simulated first to handle an encrypted file. *Ed* operates on a copy of the file it is editing; changes made to the copy have no effect on the file until a *w* (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*. There is only one buffer.

Red is a restricted version of *ed*. It will only allow editing of files in the current directory. It prohibits executing shell commands via *!shell command*. Attempts to bypass these restrictions result in an error message (*restricted shell*).

Both *ed* and *red* support the *fspec*(4) formatting capability. After including a format specification as the first line of *file* and invoking *ed* with your terminal in **stty –tabs** or **stty tab3** mode (see *stty*(1), the specified tab stops will automatically be used when scanning *file*. For example, if the first line of a file contained:

<:t5,10,15 s72:>

tab stops would be set at columns 5, 10, and 15, and a maximum line length of 72 would be imposed. NOTE: while inputting text, tab characters when typed are expanded to every eighth column as is the default.

Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be in *input mode*. In this mode, *no* commands are recognized; all input is merely collected. Input mode is left by typing a period (.) alone at the beginning of a line.

Ed supports a limited form of *regular expression* notation; regular expressions are used in addresses to specify lines and in some commands (e.g., *s*) to specify portions of a line that are to be substituted. A regular expression (RE) specifies a set of character strings. A member of this set of strings is said to be *matched* by the RE. The REs allowed by *ed* are constructed as follows:

The following *one-character RE*s match a *single* character:

1.1 An ordinary character (*not* one of those discussed in 1.2 below) is a one-character RE that matches itself.

1.2 A backslash (\) followed by any special character is a one-character RE that matches the special character itself. The special characters are:

a. ., *, [, and \ (period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets ([]; see 1.4 below).

b. ^ (caret or circumflex), which is special at the *beginning* of an *entire* RE (see 3.1 and 3.2 below), or when it immediately follows the left of a pair of square brackets ([]) (see 1.4 below).

c. $ (currency symbol), which is special at the *end* of an entire RE (see 3.2 below).

d. The character used to bound (i.e., delimit) an entire RE, which is special for that RE (for example, see how slash (/) is used in the *g* command, below.)

1.3 A period (.) is a one-character RE that matches any character except new-line.

1.4 A non-empty string of characters enclosed in square brackets ([]) is a one-character RE that matches *any one* character in that string. If, however, the first character of the string is a circumflex (^), the one-character RE matches any character *except* new-line and the remaining characters in the string. The ^ has this special meaning *only* if it occurs first in the string. The minus (-) may be used to indicate a range of consecutive ASCII characters; for example, **[0-9]** is equivalent to **[0123456789]**. The - loses this special meaning if it occurs first (after an initial ^, if any) or last in the string. The right square bracket (]) does not terminate such a string when it is the first character within it (after an initial ^, if any); e.g., **[]a-f]** matches either a right square bracket (]) or one of the letters **a** through **f** inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

The following rules may be used to construct *RE*s from one-character REs:

2.1 A one-character RE is a RE that matches whatever the one-character RE matches.

2.2 A one-character RE followed by an asterisk (*) is a RE that matches *zero* or more occurrences of the one-character RE. If there is any choice, the longest leftmost string that permits a match is chosen.

2.3 A one-character RE followed by \{*m*\}, \{*m*,\}, or \{*m*,*n*\} is a RE that matches a *range* of occurrences of the one-character RE. The values of *m* and *n* must be non-negative integers less than 256; \{*m*\} matches *exactly m* occurrences; \{*m*,\} matches *at least m* occurrences; \{*m*,*n*\} matches *any number* of occurrences *between m* and *n* inclusive. Whenever a choice exists, the RE matches as many occurrences as possible.

2.4 The concatenation of REs is a RE that matches the concatenation of the strings matched by each component of the RE.

2.5 A RE enclosed between the character sequences \(and \) is a RE that matches whatever the unadorned RE matches.

2.6 The expression *n* matches the same string of characters as was matched by an expression enclosed between \(and \) *earlier* in the same RE. Here *n* is a digit; the sub-expression specified is that beginning with the *n*-th occurrence of \(counting from the left. For example, the expression **^\(.*\)\1$** matches a line consisting of two repeated appearances of the same string.

Finally, an *entire RE* may be constrained to match only an initial segment or final segment of a line (or both).

3.1 A circumflex (^) at the beginning of an entire RE constrains that RE to match an *initial* segment of a line.

3.2 A currency symbol ($) at the end of an entire RE constrains that RE to match a *final* segment of a line.

The construction ^*entire RE*$ constrains the entire RE to match the entire line.

The null RE (e.g., //) is equivalent to the last RE encountered. See also the last paragraph before *FILES* below.

To understand addressing in *ed* it is necessary to know that at any time there is a *current line*. Generally speaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each command. *Addresses* are constructed as follows:

1. The character . addresses the current line.
2. The character $ addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. '*x* addresses the line marked with the mark name character *x*, which must be a lower-case letter. Lines are marked with the *k* command described below.
5. A RE enclosed by slashes (/) addresses the first line found by searching *forward* from the line *following* the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched. See also the last paragraph before *FILES* below.
6. A RE enclosed in question marks (?) addresses the first line found by searching *backward* from the line *preceding* the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. See also the last paragraph before *FILES* below.
7. An address followed by a plus sign (+) or a minus sign (−) followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. The plus sign may be omitted.
8. If an address begins with + or −, the addition or subtraction is taken with respect to the current line; e.g, −5 is understood to mean .−5.
9. If an address ends with + or −, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of rule 8 immediately above, the address − refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character ^ in addresses is entirely equivalent to −.) Moreover, trailing + and − characters have a cumulative effect, so −− refers to the current line less 2.
10. For convenience, a comma (,) stands for the address pair 1,$, while a semicolon (;) stands for the pair .,$.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma (,). They may also be separated by a semicolon (;). In the latter case, the current line (.) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches (see rules 5. and 6. above). The second address of any two-address sequence must correspond to a line that follows, in the buffer, the line corresponding to the first address.

In the following list of *ed* commands, the default addresses are shown in parentheses. The parentheses are *not* part of the address; they show that the given addresses are the default.

It is generally illegal for more than one command to appear on a line. However, any command (except *e*, *f*, *r*, or *w*) may be suffixed by **l**, **n**, or **p** in which case the current line is either listed, numbered or printed, respectively, as discussed below under the *l*, *n*, and *p* commands.

(.)a
<text>
.

The *a*ppend command reads the given text and appends it after the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command: it causes the "appended" text to be placed at the beginning of the buffer. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

(.)c
<text>
.

The *c*hange command deletes the addressed lines, then accepts input text that replaces these lines; . is left at the last line input, or, if there were none, at the first line that was not deleted.

(.,.)d

The *d*elete command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

e *file*

The *e*dit command causes the entire contents of the buffer to be deleted, and then the named file to be read in; . is set to the last line of the buffer. If no file name is given, the currently-remembered file name, if any, is used (see the *f* command). The number of characters read is typed; *file* is remembered for possible use as a default file name in subsequent *e*, *r*, and *w* commands. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. Such a shell command is *not* remembered as the current file name. See also *DIAGNOSTICS* below.

E *file*

The *E*dit command is like *e*, except that the editor does not check to see if any changes have been made to the buffer since the last *w* command.

f *file*

If *file* is given, the *f*ile-name command changes the currently-remembered file name to *file*; otherwise, it prints the currently-remembered file name.

(1,$)g/*RE***/***command list*

In the *g*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, the given *command list* is executed with . initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multi-line list except the last line must be ended with a \; *a*, *i*, and *c* commands and associated input are permitted. The . terminating input mode may be omitted if it would be the last line of the *command list*. An empty *command list* is equivalent to the *p* command. The *g*, *G*, *v*, and *V* commands are *not* permitted in the *command list*. See also *BUGS* and the last paragraph before *FILES* below.

(1,$)G/*RE***/**

In the interactive *G*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, that line is printed, . is changed to that line, and any *one* command (other than one of the *a*, *c*, *i*, *g*, *G*, *v*, and *V* commands) may be input and is executed. After the execution of that command, the next marked line is printed, and so on; a new-line acts as a null command; an **&** causes the re-execution of the most recent command executed within the current invocation of *G*. Note that the commands input as part of the execution of the *G* command may address and affect *any* lines in the buffer. The *G* command can be terminated by an interrupt signal (ASCII DEL or BREAK).

h

The *h*elp command gives a short error message that explains the reason for the most recent **?** diagnostic.

H

The *H*elp command causes *ed* to enter a mode in which error messages are printed for all subsequent **?** diagnostics. It will also explain the previous **?** if there was one. The *H* command alternately turns this mode on and off; it is initially off.

(.)i
<text>
.

The *i*nsert command inserts the given text before the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. This command differs from the *a* command only in the placement of the input text. Address 0 is not legal for this command. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

(.,.+1)j

The *j*oin command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given, this command does nothing.

(.)k*x*

The mar*k* command marks the addressed line with name *x*, which must be a lower-case letter. The address *'x* then addresses this line; . is unchanged.

(.,.)l

The *l*ist command prints the addressed lines in an unambiguous way: a few non-printing characters (e.g., *tab, backspace*) are represented by (hopefully) mnemonic overstrikes. All other non-printing characters are printed in octal, and long lines are folded. An *l* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.)m*a*

The *m*ove command repositions the addressed line(s) after the line addressed by *a*. Address 0 is legal for *a* and causes the addressed line(s) to be moved to the beginning of the file. It is an error if address *a* falls within the range of moved lines; . is left at the last line moved.

(.,.)n

The *n*umber command prints the addressed lines, preceding each line by its line number and a tab character; . is left at the last line printed. The *n* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.)p

The *p*rint command prints the addressed lines; . is left at the last line printed. The *p* command may be appended to any other command other than *e*, *f*, *r*, or *w*. For example, *dp* deletes the current line and prints the new current line.

P

The editor will prompt with a * for all subsequent commands. The *P* command alternately turns this mode on and off; it is initially off.

q

The *q*uit command causes *ed* to exit. No automatic write of a file is done (but see *DIAGNOSTICS* below).

Q

The editor exits without checking if changes have been made in the buffer since the last *w* command.

(.)r *file*

The *r*ead command reads in the given file after the addressed line. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands). The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. Address 0 is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed; . is set to the last line read in. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. For example, "$r !ls" appends current directory to the end of the file being edited. Such a shell command is *not* remembered as the current file name.

(.,.)s/*RE***/***replacement***/** or
(.,.)s/*RE***/***replacement***/g** or
(.,.)s/*RE***/***replacement***/n** n = 1-512

The *s*ubstitute command searches each addressed line for an occurrence of the specified RE. In each line in which a match is found, all (non-overlapped) matched strings are replaced by the *replacement* if the global replacement indicator **g** appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. If a number n appears after the command, only the n th occurrence of the matched string on each addressed line

is replaced. It is an error for the substitution to fail on *all* addressed lines. Any character other than space or new-line may be used instead of / to delimit the RE and the *replacement*; . is left at the last line on which a substitution occurred. See also the last paragraph before *FILES* below.

An ampersand (&) appearing in the *replacement* is replaced by the string matching the RE on the current line. The special meaning of & in this context may be suppressed by preceding it by \. As a more general feature, the characters *n*, where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression of the specified RE enclosed between \(and \). When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of \(starting from the left. When the character % is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The % loses its special meaning when it is in a replacement string of more than one character or is preceded by a \.

A line may be split by substituting a new-line character into it. The new-line in the *replacement* must be escaped by preceding it by \. Such substitution cannot be done as part of a *g* or *v* command list.

(.,.)t*a*

This command acts just like the *m* command, except that a *copy* of the addressed lines is placed after address *a* (which may be 0); . is left at the last line of the copy.

u

The *u*ndo command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent *a*, *c*, *d*, *g*, *i*, *j*, *m*, *r*, *s*, *t*, *v*, *G*, or *V* command.

(1,$)v/*RE/command list*

This command is the same as the global command *g* except that the *command list* is executed with . initially set to every line that does *not* match the RE.

(1,$)V/*RE/*

This command is the same as the interactive global command *G* except that the lines that are marked during the first step are those that do *not* match the RE.

(1,$)w *file*

The *w*rite command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writable by everyone), unless your *umask* setting (see *sh*(1)) dictates otherwise. The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands); . is unchanged. If the command is successful, the number of characters written is typed. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose standard input is the addressed lines. Such a shell command is *not* remembered as the current file name.

X

A key string is demanded from the standard input. Subsequent *e*, *r*, and *w* commands will encrypt and decrypt the text with this key by the algorithm of *crypt*(1). An explicitly empty key turns off encryption.

(**$**)**=**
The line number of the addressed line is typed; **.** is unchanged by this command.

!*shell command*
The remainder of the line after the **!** is sent to the UNIX system shell (*sh*(1)) to be interpreted as a command. Within the text of that command, the unescaped character **%** is replaced with the remembered file name; if a **!** appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, **!!** will repeat the last shell command. If any expansion is performed, the expanded line is echoed; **.** is unchanged.

(**.+1**)<new-line>
An address alone on a line causes the addressed line to be printed. A new-line alone is equivalent to **.+1p**; it is useful for stepping forward through the buffer.

If an interrupt signal (ASCII DEL or BREAK) is sent, *ed* prints a **?** and returns to *its* command level.

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per file name, and 128K characters in the buffer. The limit on the number of lines depends on the amount of user memory: each line takes 1 word.

When reading a file, *ed* discards ASCII NUL characters and all characters after the last new-line. Files (e.g., **a.out**) that contain characters not in the ASCII set (bit 8 on) cannot be edited by *ed*.

If the closing delimiter of a RE or of a replacement string (e.g., /) would be the last character before a new-line, that delimiter may be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

s/s1/s2	s/s1/s2/p
g/s1	g/s1/p
?s1	?s1?

FILES

/tmp/e#	temporary; # is the process number.
ed.hup	work is saved here if the terminal is hung up.

DIAGNOSTICS

?	for command errors.
?*file*	for an inaccessible file.
	(use the *h*elp and *H*elp commands for detailed explanations).

If changes have been made in the buffer since the last *w* command that wrote the entire buffer, *ed* warns the user if an attempt is made to destroy *ed*'s buffer via the *e* or *q* commands. It prints **?** and allows one to continue editing. A second *e* or *q* command at this point will take effect. The **−** command-line option inhibits this feature.

SEE ALSO

crypt(1).
grep(1), sed(1), sh(1), stty(1) in the *3B2 Computer System User Reference Manual.*
fspec(4), regexp(5) in the *3B2 Computer System Programmer Reference Manual.*

BUGS

A *!* command cannot be subject to a *g* or a *v* command.
The *!* command and the **!** escape from the *e*, *r*, and *w* commands cannot be used if the the editor is invoked from a restricted shell (see *sh*(1)).
The sequence **\n** in a RE does not match a new-line character.
The *l* command mishandles DEL.
Files encrypted directly with the *crypt*(1) command with the null key cannot be edited.
Characters are masked to 7 bits on input.
If the editor input is coming from a command file (i.e., ed file < ed-cmd-file), the editor will exit at the first failure of a command that is in the command file.

NAME

edit – text editor (variant of ex for casual users)

SYNOPSIS

edit [**–r**] name ...

DESCRIPTION

Edit is a variant of the text editor *ex* recommended for new or casual users who wish to use a command-oriented editor.

–r Recover file after an editor or system crash.

The following brief introduction should help you get started with *edit*. If you are using a CRT terminal you may want to learn about the display editor *vi*.

To edit the contents of an existing file you begin with the command "edit name" to the shell. *Edit* makes a copy of the file which you can then edit, and tells you how many lines and characters are in the file. To create a new file, just make up a name for the file and try to run *edit* on it; you will cause an error diagnostic, but do not worry.

Edit prompts for commands with the character ':', which you should see after starting the editor. If you are editing an existing file, then you will have some lines in *edit's* buffer (its name for the copy of the file you are editing). Most commands to *edit* use its "current line" if you do not tell them which line to use. Thus if you say **print** (which can be abbreviated **p**) and hit carriage return (as you should after all *edit* commands) this current line will be printed. If you **delete** (**d**) the current line, *edit* will print the new current line. When you start editing, *edit* makes the last line of the file the current line. If you **delete** this last line, then the new last line becomes the current one. In general, after a **delete,** the next line in the file becomes the current line. (Deleting the last line is a special case.)

If you start with an empty file or wish to add some new lines, then the **append** (**a**) command can be used. After you give this command (typing a carriage return after the word **append**) *edit* will read lines from your terminal until you give a line consisting of just a ".", placing these lines after the current line. The last line you type then becomes the current line. The command **insert** (**i**) is like **append** but places the lines you give before, rather than after, the current line.

Edit numbers the lines in the buffer, with the first line having number 1. If you give the command "1" then *edit* will type this first line. If you then give the command **delete** *edit* will delete the first line, line 2 will become line 1, and *edit* will print the current line (the new line 1) so you can see where you are. In general, the current line will always be the last line affected by a command.

You can make a change to some text within the current line by using the **substitute** (**s**) command. You say "s/*old*/*new*/" where *old* is replaced by the old characters you want to get rid of and *new* is the new characters you want to replace it with.

The command **file** (**f**) will tell you how many lines there are in the buffer you are editing and will say "[Modified]" if you have changed it. After modifying a file you can put the buffer text back to replace the file by giving a **write** (**w**) command. You can then leave the editor by issuing a **quit** (**q**) command. If you run *edit* on a file, but do not change it, it is not necessary (but does no harm) to **write** the file back. If you try to **quit** from *edit* after modifying the buffer without writing it out, you will be warned that there has been "No **write** since last change" and *edit* will await another command. If you wish not to **write** the buffer out then you can issue another **quit** command. The buffer is then irretrievably discarded, and you return to the shell.

By using the **delete** and **append** commands, and giving line numbers to see lines in the file you can make any changes you desire. You should learn at least a few more things, however, if you are to use *edit* more than a few times.

The **change** (**c**) command will change the current line to a sequence of lines you supply (as in **append** you give lines up to a line consisting of only a "."). You can tell **change** to change more than one line by giving the line numbers of the lines you want to change, i.e., "3,5change". You can print lines this way too. Thus "1,23p" prints the first 23 lines of the file.

The **undo** (**u**) command will reverse the effect of the last command you gave which changed the buffer. Thus if you give a **substitute** command which does not do what you want, you can say **undo** and the old contents of the line will be restored. You can also **undo** an **undo** command so that you can continue to change your mind. *Edit* will give you a warning message when commands you do affect more than one line of the buffer. If the amount of change seems unreasonable, you should consider doing an *undo* and looking to see what happened. If you decide that the change is ok, then you can *undo* again to get it back. Note that commands such as *write* and *quit* cannot be undone.

To look at the next line in the buffer you can just hit carriage return. To look at a number of lines hit ^D (control key and, while it is held down D key, then let up both) rather than carriage return. This will show you a half screen of lines on a CRT or 12 lines on a hardcopy terminal. You can look at the text around where you are by giving the command "z.". The current line will then be the last line printed; you can get back to the line where you were before the "z." command by saying "''". The **z** command can also be given other following characters "z−" prints a screen of text (or 24 lines) ending where you are; "z+" prints the next screenful. If you want less than a screenful of lines, type in "z.12" to get 12 lines total. This method of giving counts works in general; thus you can delete 5 lines starting with the current line with the command "delete 5".

To find things in the file, you can use line numbers if you happen to know them; since the line numbers change when you insert and delete lines this is somewhat unreliable. You can search backwards and forwards in the file for strings by giving commands of the form **/text/** to search forward for *text* or **?text?** to search backward for *text*. If a search reaches the end of the file without finding the text it wraps, end around, and continues to search back to the line where you are. A useful feature here is a search of the form **/^text/** which searches for *text* at the beginning of a line. Similarly **/text$/** searches for *text* at the end of a line. You can leave off the trailing / or ? in these commands.

The current line has a symbolic name "."; this is most useful in a range of lines as in ".,$print" which prints the rest of the lines in the file. To get to the last line in the file you can refer to it by its symbolic name "$". Thus the command "$ delete" or "$d" deletes the last line in the file, no matter which line was the current line before. Arithmetic with line references is also possible. Thus the line "$−5" is the fifth before the last, and ".+20" is 20 lines after the present.

You can find out which line you are at by doing ".=". This is useful if you wish to move or copy a section of text within a file or between files. Find out the first and last line numbers you wish to copy or move (say 10 to 20). For a move you can then say "10,20delete a" which deletes these lines from the file and places them in a buffer named *a*. *Edit* has 26 such buffers named *a* through *z*. You can later get these lines back by doing "put a" to put the contents of buffer *a* after the current line. If you want to move or copy these lines between files you can give an **edit** (**e**) command after copying the lines,

following it with the name of the other file you wish to edit, i.e., "edit chapter2". By changing *delete* to *yank* above you can get a pattern for copying lines. If the text you wish to move or copy is all within one file then you can just say "10,20move $" for example. It is not necessary to use named buffers in this case (but you can if you wish).

SEE ALSO

ex(1), vi(1).

NAME

ex – text editor

SYNOPSIS

ex [**–**] [**–v**] [**–t** *tag*] [**–r**] [**–R**] [**+***command*] [**–x**] name ...

DESCRIPTION

Ex is the root of a family of editors: *ex* and *vi*. *Ex* is a superset of *ed*, with the most notable extension being a display editing facility. Display based editing is the focus of *vi*.

If you have a CRT terminal, you may wish to use a display based editor; in this case see *vi*(1), which is a command which focuses on the display editing portion of *ex*.

FOR ED USERS

If you have used *ed* you will find that *ex* has a number of new features useful on CRT terminals. Intelligent terminals and high speed terminals are very pleasant to use with *vi*. Generally, the editor uses far more of the capabilities of terminals than *ed* does, and uses the terminal capability data base *terminfo*(4) and the type of the terminal you are using from the variable TERM in the environment to determine how to drive your terminal efficiently. The editor makes use of features such as insert and delete character and line in its **visual** command (which can be abbreviated **vi**) and which is the central mode of editing when using *vi*(1).

Ex contains a number of new features for easily viewing the text of the file. The **z** command gives easy access to windows of text. Hitting ^D causes the editor to scroll a half-window of text and is more useful for quickly stepping through a file than just hitting return. Of course, the screen-oriented **visual** mode gives constant access to editing context.

Ex gives you more help when you make mistakes. The **undo** (**u**) command allows you to reverse any single change which goes astray. *Ex* gives you a lot of feedback, normally printing changed lines, and indicates when more than a few lines are affected by a command so that it is easy to detect when a command has affected more lines than it should have.

The editor also normally prevents overwriting existing files unless you edited them so that you do not accidentally clobber with a *write* a file other than the one you are editing. If the system (or editor) crashes, or you accidentally hang up the telephone, you can use the editor **recover** command to retrieve your work. This will get you back to within a few lines of where you left off.

Ex has several features for dealing with more than one file at a time. You can give it a list of files on the command line and use the **next** (**n**) command to deal with each in turn. The **next** command can also be given a list of file names, or a pattern as used by the shell to specify a new set of files to be dealt with. In general, file names in the editor may be formed with full shell metasyntax. The metacharacter '%' is also available in forming file names and is replaced by the name of the current file.

For moving text between files and within a file the editor has a group of buffers, named *a* through *z*. You can place text in these named buffers and carry it over when you edit another file.

There is a command **&** in *ex* which repeats the last **substitute** command. In addition there is a confirmed substitute command. You give a range of substitutions to be done and the editor interactively asks whether each substitution is desired.

It is possible to ignore case of letters in searches and substitutions. *Ex* also allows regular expressions which match words to be constructed. This is convenient, for example, in searching for the word "edit" if your document also contains the word "editor."

Ex has a set of *options* which you can set to tailor it to your liking. One option which is very useful is the *autoindent* option which allows the editor to automatically supply leading white space to align text. You can then use the ^D key as a backtab and space and tab forward to align new code easily.

Miscellaneous new useful features include an intelligent **join** (**j**) command which supplies white space between joined lines automatically, commands < and > which shift groups of lines, and the ability to filter portions of the buffer through commands such as *sort*.

INVOCATION OPTIONS

The following invocation options are interpreted by *ex*:

−	Suppress all interactive-user feedback. This is useful in processing editor scripts.
−v	Invokes *vi*
−t *tagfR*	Edit the file containing the *tag* and position the editor at its definition.
−r *file*	Recover *file* after an editor or system crash. If *file* is not specified a list of all saved files will be printed.
−R	*Readonly* mode set, prevents accidentally overwriting the file.
+*command*	Begin editing by executing the specified editor search or positioning *command*.
−x	Encryption mode; a key is prompted for allowing creation or editing of an encrypted file.

The *name* argument indicates files to be edited.

Ex States

Command	Normal and initial state. Input prompted for by :. Your kill character cancels partial command.
Insert	Entered by **a i** and **c**. Arbitrary text may be entered. Insert is normally terminated by line having only . on it, or abnormally with an interrupt.
Visual	Entered by **vi**, terminates with **Q** or ^\.

Ex command names and abbreviations

abbrev	**ab**	next	**n**	unabbrev	**una**
append	**a**	number	**nu**	undo	**u**
args	**ar**			unmap	**unm**
change	**c**	preserve	**pre**	version	**ve**
copy	**co**	print	**p**	visual	**vi**
delete	**d**	put	**pu**	write	**w**
edit	**e**	quit	**q**	xit	**x**
file	**f**	read	**re**	yank	**ya**
global	**g**	recover	**rec**	window	**z**
insert	**i**	rewind	**rew**	escape	**!**
join	**j**	set	**se**	lshift	**<**
list	**l**	shell	**sh**	print next	CR
map		source	**so**	resubst	**&**
mark	**ma**	stop	**st**	rshift	**>**
move	**m**	substitute	**s**	scroll	**^D**

Ex Command Addresses

n	line *n*	*/pat*	next with *pat*
.	current	**?***pat*	previous with *pat*
$	last	*x*-*n*	*n* before *x*
+	next	*x*,*y*	*x* through *y*
−	previous	*'x*	marked with *x*
+*n*	*n* forward	*''*	previous context
%	1,$		

Initializing options

EXINIT	place **set**'s here in environment var.
$HOME/.exrc	editor initialization file
./.exrc	editor initialization file
set *x*	enable option
set no*x*	disable option
set *x*=*val*	give value *val*
set	show changed options
set all	show all options
set *x*?	show value of option *x*

Most useful options

autoindent	ai	supply indent
autowrite	aw	write before changing files
ignorecase	ic	in scanning
list		print ^I for tab, $ at end
magic		. [* special in patterns
number	nu	number lines
paragraphs	para	macro names which start ...
redraw		simulate smart terminal
scroll		command mode lines
sections	sect	macro names ...
shiftwidth	sw	for < >, and input ^D
showmatch	sm	to) and } as typed
showmode	smd	show insert mode in *vi*
slowopen	slow	stop updates during insert
window		visual mode lines
wrapscan	ws	around end of buffer?
wrapmargin	wm	automatic line splitting

Scanning pattern formation

^	beginning of line
$	end of line
.	any character
\<	beginning of word
\>	end of word
[*str*]	any char in *str*
[↑*str*]	... not in *str*
[*x*−*y*]	... between *x* and *y*
*	any number of preceding

AUTHOR

Vi and *ex* are based on software developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

/usr/lib/ex?.?strings	error messages
/usr/lib/ex?.?recover	recover command
/usr/lib/ex?.?preserve	preserve command
/usr/lib/*/*	describes capabilities of terminals
$HOME/.exrc	editor startup file
./.exrc	editor startup file
/tmp/Ex*nnnnn*	editor temporary
/tmp/Rx*nnnnn*	named buffer temporary
/usr/preserve	preservation directory

SEE ALSO

ed(1), vi(1).
awk(1), edit(1), grep(1), sed(1) in the *3B2 Computer System User Reference Manual.*
curses(3X), term(4), terminfo(4) in the *3B2 Computer System Programmer Reference Manual.*

BUGS

The *undo* command causes all marks to be lost on lines changed and then restored if the marked lines were changed.

Undo never clears the buffer modified condition.

The *z* command prints a number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors do not print a name if the command line '−' option is used.

There is no easy way to do a single scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files and cannot appear in resultant files.

NAME

makekey – generate encryption key

SYNOPSIS

/usr/lib/makekey

DESCRIPTION

Makekey improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (i.e., to require a substantial fraction of a second).

The first eight input bytes (the *input key*) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, ., /, and upper- and lower-case letters. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key*.

The transformation performed is essentially the following: the salt is used to select one of 4,096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but broken in 4,096 different ways. Using the *input key* as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 *output key* bits in the result.

Makekey is intended for programs that perform encryption (e.g., *ed*(1) and *crypt*(1)). Usually, its input and output will be pipes.

SEE ALSO

crypt(1), ed(1).
passwd(4) in the *3B2 Computer System Programmer Reference Manual.*

NAME

vi – screen-oriented (visual) display editor based on ex

SYNOPSIS

vi [**−t** *tag*] [**−r** *file*] [**−w***n*] [**−x**] [**−R**] [**+***command*] name ...

view [**-t** *tag*] [**−r** *file*] [**−w***n*] [**−x**] [**−R**] [**+***command*] name ...

vedit [**−t** *tag*] [**−r** *file*] [**−w***n*] [**−x**] [**−R**] [**+***command*] name ...

DESCRIPTION

Vi (visual) is a display-oriented text editor based on an underlying line editor *ex*(1). It is possible to use the command mode of *ex* from within *vi* and vice-versa.

When using *vi*, changes you make to the file are reflected in what you see on your terminal screen. The position of the cursor on the screen indicates the position within the file.

INVOCATION

The following invocation options are interpreted by *vi*:

−t *tag*	Edit the file containing the *tag* and position the editor at its definition.
−r*file*	Recover *file* after an editor or system crash. If *file* is not specified a list of all saved files will be printed.
−w*n*	Set the default window size to *n*. This is useful when using the editor over a slow speed line.
−x	Encryption mode; a key is prompted for allowing creation or editing of an encrypted file.
−R	Read only mode; the **readonly** flag is set, preventing accidental overwriting of the file.
+*command*	The specified *ex* command is interpreted before editing begins.

The *name* argument indicates files to be edited.

The *view* invocation is the same as *vi* except that the **readonly** flag is set.

The *vedit* invocation is intended for beginners. The **report** flag is set to 1, and the **showmode** and **novice** flags are set. These defaults make it easier to get started learning the editor.

"VI MODES"

Command	Normal and initial mode. Other modes return to command mode upon completion. ESC (escape) is used to cancel a partial command.
Input	Entered by the following options: **a i A I o O c C s S R**. Arbitrary text may then be entered. Input mode is normally terminated with ESC character, or abnormally with interrupt.
Last line	Reading input for **:** **/** **?** or **!**; terminate with CR to execute, interrupt to cancel.

COMMAND SUMMARY

Sample commands

← ↓ ↑ →	arrow keys move the cursor
h j k l	same as arrow keys
i*text*ESC	insert text *abc*
cw*new*ESC	change word to *new*

ea*s*ESC	pluralize word
x	delete a character
dw	delete a word
dd	delete a line
3dd	... 3 lines
u	undo previous change
ZZ	exit vi, saving changes
:q!CR	quit, discarding changes
*/**text**CR*	search for *text*
^U ^D	scroll up or down
:*ex cmd***CR**	any ex or ed command

Counts before vi commands

Numbers may be typed as a prefix to some commands. They are interpreted in one of these ways.

line/column number	**z G |**
scroll amount	**^D ^U**
repeat effect	most of the rest

Interrupting, canceling

ESC	end insert or incomplete cmd
^?	(delete or rubout) interrupts
^L	reprint screen if ^? scrambles it
^R	reprint screen if ^L is → key

File manipulation

:wCR	write back changes
:qCR	quit
:q!CR	quit, discard changes
:e *name***CR**	edit file *name*
:e!CR	reedit, discard changes
:e + *name*CR	edit, starting at end
:e +*n*CR	edit starting at line *n*
:e #CR	edit alternate file
	synonym for **:e #**
:w *name***CR**	write file *name*
:w! *name***CR**	overwrite file *name*
:shCR	run shell, then return
:!*cmd***CR**	run *cmd*, then return
:n CR	edit next file in arglist
:n *args***CR**	specify new arglist
^G	show current file and line
:ta *tag***CR**	to tag file entry *tag*
^]	**:ta**, following word is *tag*

In general, any *ex* or *ed* command (such as *substitute* or *global*) may be typed, preceded by a colon and followed by a **CR**.

Positioning within file

^F	forward screen
^B	backward screen
^D	scroll down half screen
^U	scroll up half screen
G	go to specified line (end default)
/pat	next line matching *pat*
?*pat*	prev line matching *pat*
n	repeat last / or ?
N	reverse last / or ?
/pat/+n	nth line after *pat*
?*pat***?−***n*	nth line before *pat*
]]	next section/function
[[	previous section/function
(	beginning of sentence
)	end of sentence
{	beginning of paragraph
}	end of paragraph
%	find matching ( ) { or }

Adjusting the screen

^L	clear and redraw
^R	retype, eliminate @ lines
zCR	redraw, current at window top
z−CR	... at bottom
z.CR	... at center
/pat/z−CR	*pat* line at bottom
z*n***.CR**	use *n* line window
^E	scroll window down 1 line
^Y	scroll window up 1 line

Marking and returning

``	move cursor to previous context
''	... at first non-white in line
m*x*	mark current position with letter *x*
`*x*	move cursor to mark *x*
'*x*	... at first non-white in line

Line positioning

H	top line on screen
L	last line on screen
M	middle line on screen
+	next line, at first non-white
−	previous line, at first non-white
CR	return, same as +
↓ or **j**	next line, same column
↑ or **k**	previous line, same column

Character positioning

^	first non white
0	beginning of line
$	end of line
h or →	forward
l or ←	backwards
^H	same as ←
space	same as →
f*x*	find *x* forward
F*x*	f backward
t*x*	upto *x* forward
T*x*	back upto *x*
;	repeat last **f F t** or **T**
,	inverse of **;**
|	to specified column
%	find matching **(** {**)** or **}**

Words, sentences, paragraphs

w	word forward
b	back word
e	end of word
)	to next sentence
}	to next paragraph
(	back sentence
{	back paragraph
W	blank delimited word
B	back W
E	to end of W

Corrections during insert

^H	erase last character
^W	erase last word
erase	your erase, same as **^H**
kill	your kill, erase input this line
****	quotes **^H**, your erase and kill
ESC	ends insertion, back to command
^?	interrupt, terminates insert
^D	backtab over *autoindent*
↑^D	kill *autoindent*, save for next
0^D	... but at margin next also
^V	quote non-printing character

Insert and replace

a	append after cursor
i	insert before cursor
A	append at end of line
I	insert before first non-blank
o	open line below
O	open above
r*x*	replace single char with *x*
R*text***ESC**	replace characters

Operators

Operators are followed by a cursor motion, and affect all text that would have been moved over. For example, since **w** moves over a word, **dw** deletes the word that would be moved over. Double the operator, e.g., **dd** to affect whole lines.

d	delete
c	change
y	yank lines to buffer
<	left shift
>	right shift
!	filter through command

Miscellaneous Operations

C	change rest of line (**c$**)
D	delete rest of line (**d$**)
s	substitute chars (**cl**)
S	substitute lines (**cc**)
J	join lines
x	delete characters (**dl**)
X	... before cursor (**dh**)
Y	yank lines (**yy**)

Yank and Put

Put inserts the text most recently deleted or yanked. However, if a buffer is named, the text in that buffer is put instead.

p	put back text after cursor
P	put before cursor
"*x*p	put from buffer *x*
"*x*y	yank to buffer *x*
"*x*d	delete into buffer *x*

Undo, Redo, Retrieve

u	undo last change
U	restore current line
.	repeat last change
"*d*p	retrieve *d*'th last delete

AUTHOR

Vi and *ex* were developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

SEE ALSO

ex (1).
3B2 Computer System Editing Utilities Guide.

BUGS

Software tabs using ^T work only immediately after the *autoindent*.

Left and right shifts on intelligent terminals do not make use of insert and delete character operations in the terminal.

Index

C

E

H

M

S

V

X